Arctic Region Supercomputing Center

MPI-IO and MPI-2 One-Sided Communication

Tom Logan

HPC Specialist

Arctic Region Supercomputing Center

HPC Modernization Program

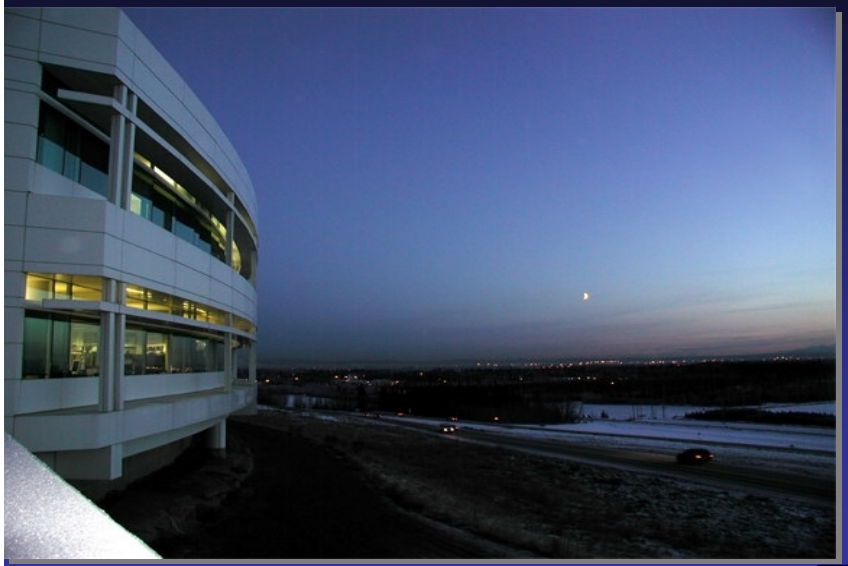
UAF University of Alaska Fairbanks

Contents

- ARSC and Iceberg
- Global Tsunami Code in CAF
- Conversion to MPI-2
- MPI-IO Tests
- MPI2 One-Sided Tests
- Global Tsunami Code in MPI
- Conclusions

"Mistakes - It could be that the purpose of your life is only to serve as a warning to others"

Arctic Region Supercomputing Center



What we do:

- High performance computing, university owned and operated center
- DoD funded through HPCMP
- Provide HPC resources and support
- Conduct research locally, globally

Who we are:

Committed to helping scientists seek understanding of our past, present and future by applying computational technology to advance discovery, analysis and prediction

DoD HPC Modernization Program

UAF University of Alaska Fairbanks

IBM System

Iceberg

- 92 p655+ servers, each with 8 processors and 16 GB of memory per server
- Four p655 I/O servers and two interactive servers, each with 8 processors and 16 GB of memory per server
- 25 TB GPFS disk
- Full Federation-2 interconnect between the components

Global Tsunami Code

- **Shallow Water Non-Linear Equation Model**
  - models tsunami propagation
  - incorporates a moving wet-dry boundary
- **For December 2004 Indonesia Tsunami**
  - 1-minute data posting (GEBCO) over nearly entire globe (220 Million grid cells)
  - 60 Cray X1 MSP's 20 hours to complete a 30 hour simulation

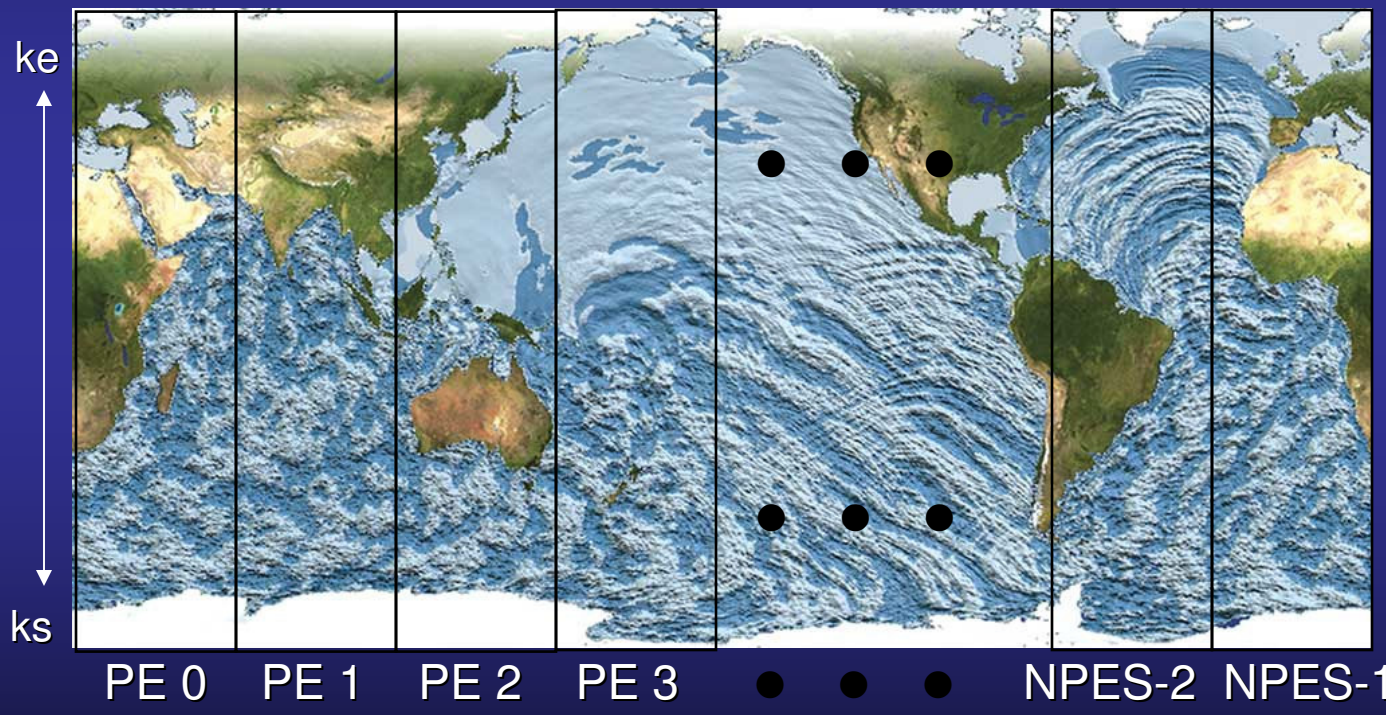
Co-Array Fortran

- **Natural Mapping to CAF**
  - Implementation took <1 day for novice CAF programmer
  - Very Simple access to off processor/node memory
  - Example declaration and access:

    REAL, DIMENSION(1:JE,1:KE)[*]:: SLO,SLN

    SLN(JE,K) = SLO(JS,K)[NEXT]
- **Communication Paradigm**
  - Only E-W communications required
    - MYPE-1 is PREV; MYPE+1 is NEXT
    - First and Last PEs are connected (cyclical version of code)
    - Clipped North at Bering Strait; Clipped South at Antarctica

Column Data Distribution



Typical CAF IO Operation

```
! CODE EXCUTED ONLY BY SINGLE MASTER PROCESSOR
frame = M/600
do k=ks,ke
   do i=1,NPES
      do j=js,je
         if (cell(j,k)[i].eq.0) then
            zwrite(j+(i-1)*JE)=10.
         else
            zwrite(j+(i-1)*JE)=slo(j,k)[i]
         end if
      end do
   end do
   write (15,rec=frame*ke+k) (zwrite(J),J=JS,JE*NPES)
end do
```

Typical CAF Memory Access

```
DO K=KS,KE
   IF (CELL(JE,K).EQ.0.AND.CELL(JS,K)[NEXT].EQ.1) THEN
      IF ((SLO(JS,K)[NEXT] + H(JE,K)-ETO(JE,K)).GT.EPSILON) then
         D(JE,K)= SLO(JS,K)[NEXT] + H(JE,K)-ETO(JE,K)
         CELL(JE,K) = 1
         CELLX(JE,K) = 1
         IF(CELL(JS+1,K)[NEXT].EQ.1)THEN
            UO(JS,K)[NEXT]=UO(JS+1,K)[NEXT]
         ELSE
            UO(JS,K)[NEXT]=-SQRT(G*D(JE,K))
         END IF
      END IF
   END IF
 END DO
```

Motivated MPI-2 Conversion

- **CRAY X1 was retired Sept 2006**
- **New pacific Tsunami event**
  - 15 November, 2006 at Kuril Island, Russia
  - Magnitude 8.3
  - Crescent City, CA
    - two docks destroyed, one more damaged
    - measured wave height of 176 cm
- **Dr. Kowalik needed simulation!**

Code Conversion to MPI-2

- **Simplest Path to Port the code to Iceberg seemed to be**
  - Convert CAF references into MPI-2 One-Sided communication
  - Convert IO to use MPI-IO
  - Although motivated to get results, wanted to test and learn MPI-2 as well
- **Soon realized that this plan was flawed**
  - IO performance was very bad
  - Many difficulties encountered in implementation

MPI-IO Testing

Simplified IO Kernel - Serial

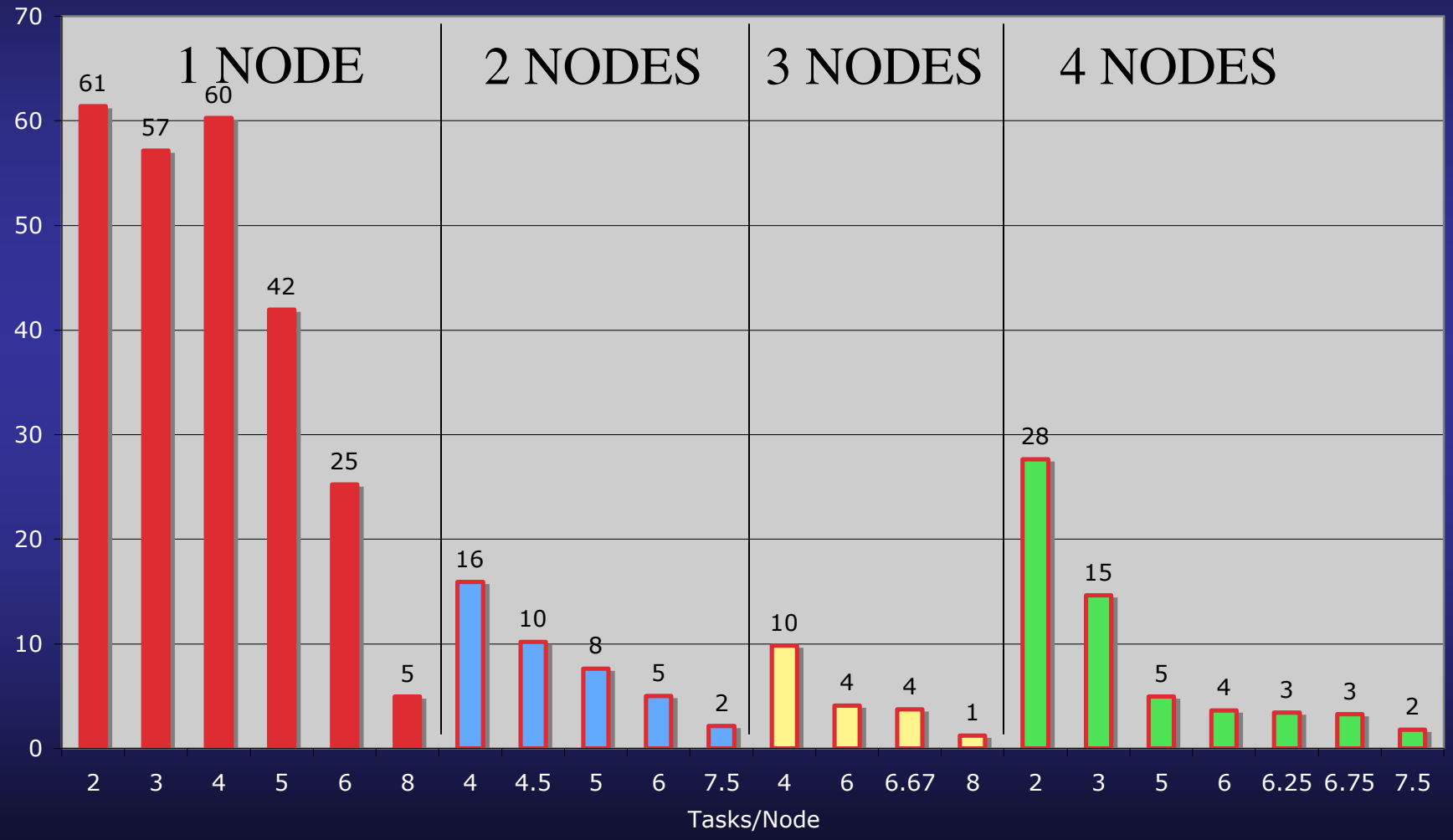
```
do cnt=1,60
   call system_clock(start,rate,max)
   do k=ks,ke
      write(15,rec=(cnt-1)*ke+k) (H(j,k),J=JS,JE)
   end do
   call system_clock(after,rate,max)
   write_time = after-start
   write_time = write_time / rate
   sum = sum + write_time
   print *,'Frame ',cnt,' time was ',write_time,'seconds'
end do
print *,'total time for writes',sum
```

- **AVERAGE BANDWIDTH 134 MB/sec**

1st Version - WRITE_AT

```
myoffset = mype * JE * data_size
linesize = JE * data_size * NPES
do cnt=1,60
   write_time = MPI_WTIME()
   do k=ks,ke
      aline = H(:,k)
      call MPI_FILE_WRITE_AT(file,myoffset,aline,JE,MPI_REAL,stat,ierr)
      myoffset = myoffset + linesize
   end do
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
   write_time = MPI_WTIME()-write_time
   sum = sum + write_time
   if (mype.eq.0) print *,'Frame ',cnt,'time was ',write_time,'seconds'
end do
call MPI_FILE_CLOSE(thefile,ierr)
if (mype.EQ.0) print *,'PE',mype,' total time for writes',sum
```

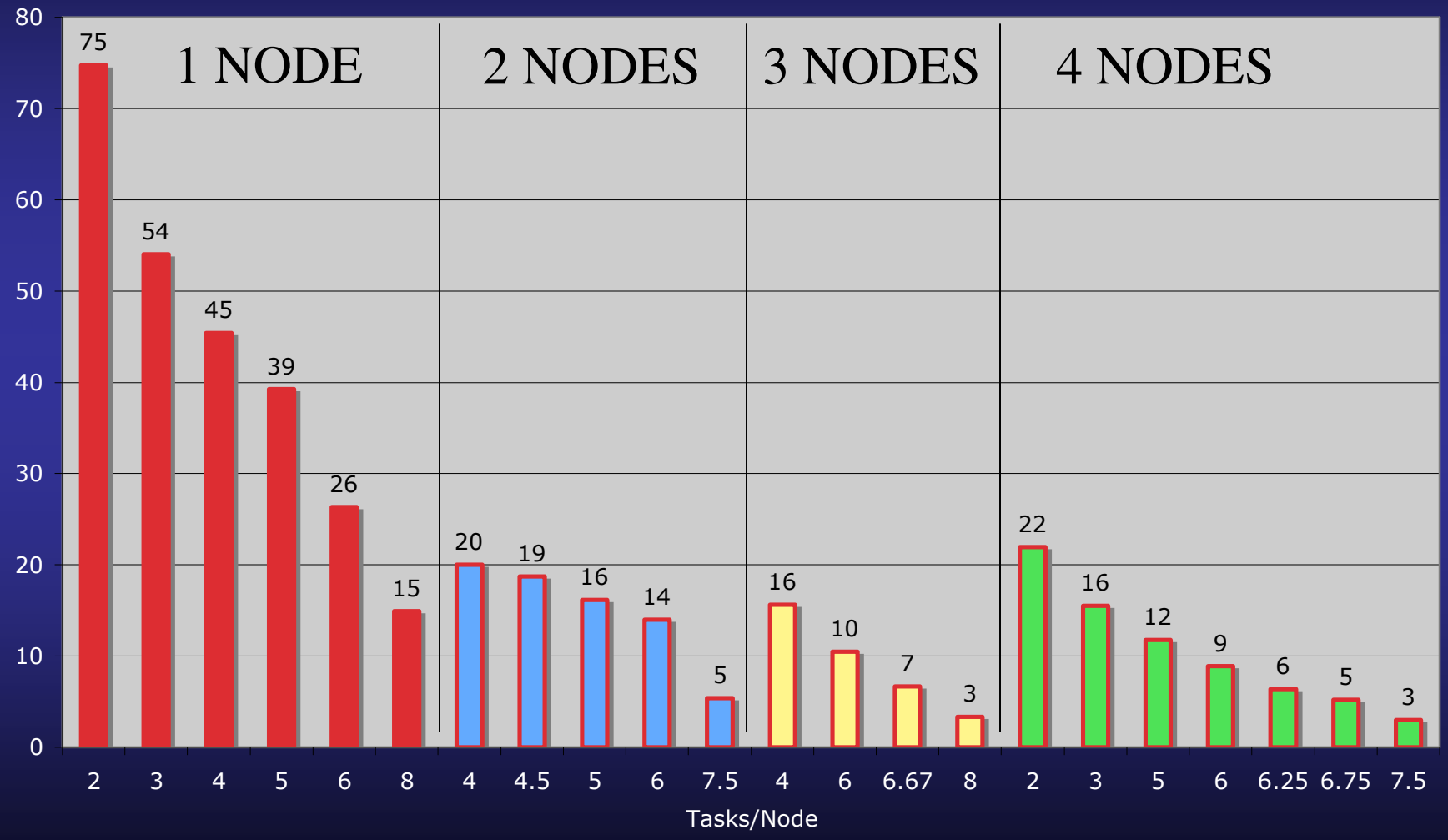
MPI IO WRITE_AT



2nd Version - WRITE_AT_ALL

```
myoffset = mype * JE * data_size
linesize = JE * data_size * NPES
do cnt=1,60
   write_time = MPI_WTIME()
   do k=ks,ke
      aline = H(:,k)
      call MPI_FILE_WRITE_AT_ALL(file,myoffset,aline,JE,MPI_REAL,...)
      myoffset = myoffset + linesize
   end do
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
   write_time = MPI_WTIME()-write_time
   sum = sum + write_time
   if (mype.eq.0) print *,'Frame ',cnt,'time was ',write_time,'seconds'
end do
call MPI_FILE_CLOSE(thefile,ierr)
if (mype.EQ.0) print *,'PE',mype,' total time for writes',sum
```

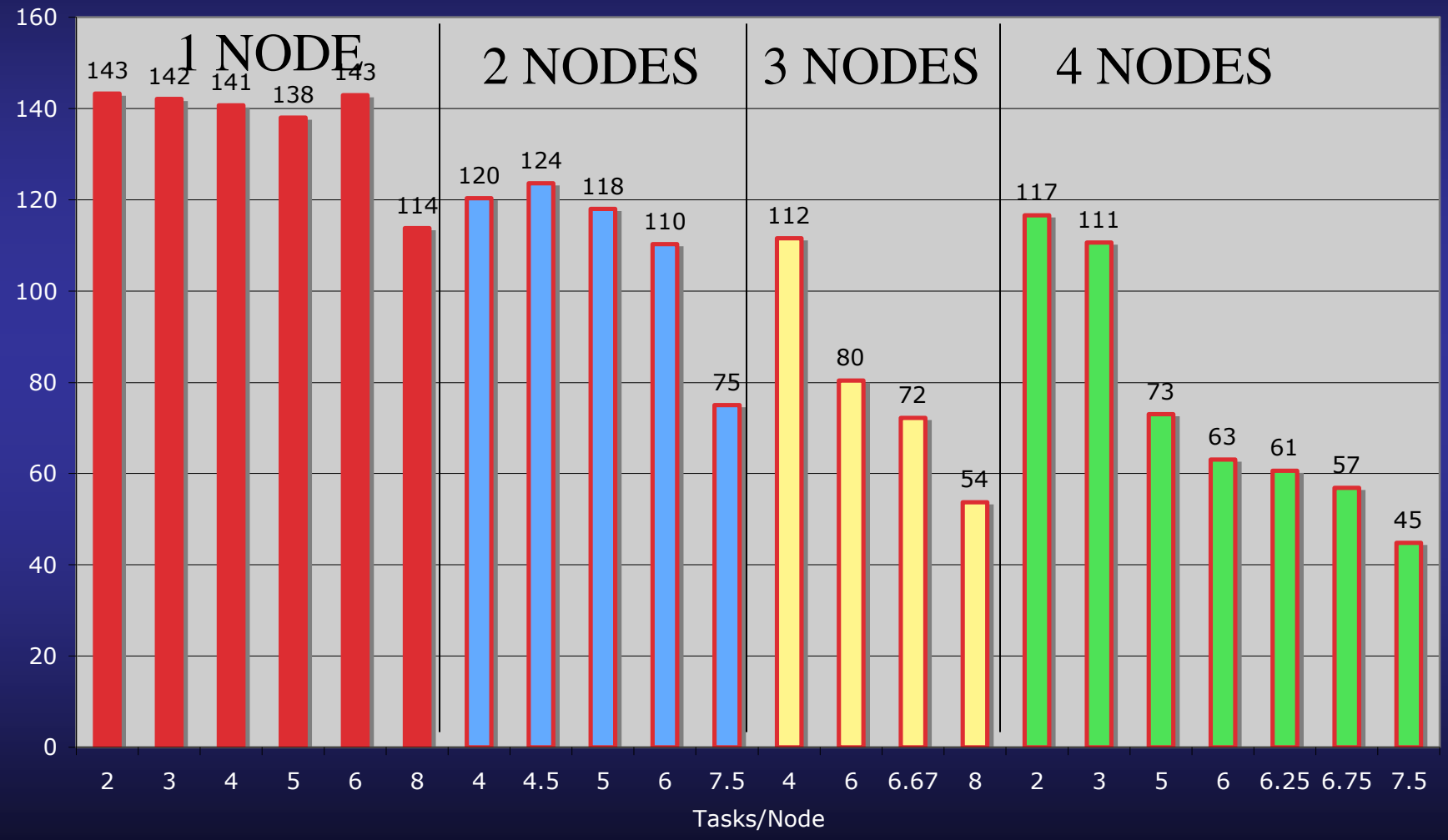
MPI IO WRITE_AT_ALL



3rd Version - Send/Recv

```
do cnt=1,60
  write_time = MPI_WTIME()
  do k=ks,ke
    if (mype.eq.0) then
       global_line(1:JE)=H(:,k)
       do pe = 1,NPES-1
         offset = pe * JE
         call mpi_recv(global_line(offset),JE,MPI_REAL,pe,k,...)
       end do
       write(15,rec=(cnt-1)*ke+k) (global_line(j),J=JS,JE*NPES)
    else
       aline = H(:,k)
       call mpi_send(aline,JE,MPI_REAL,0,k,MPI_COMM_WORLD,ierr)
    end if
  end do
  call MPI_BARRIER(MPI_COMM_WORLD,ierr)
< calc write_time, sum >
end do
```

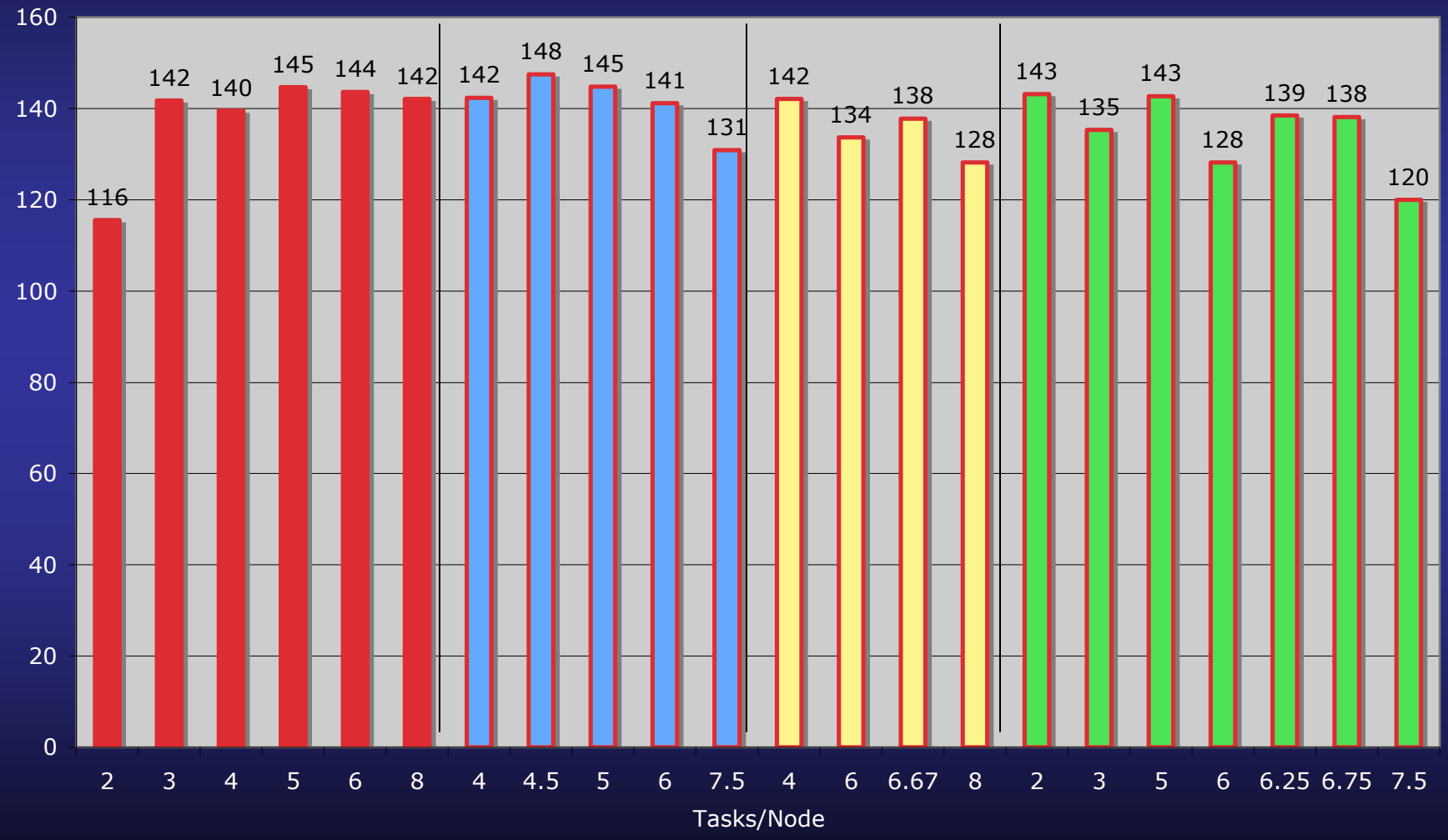
Send/Recv MPI collection



4th Version - Gather

```
do cnt=1,60
  write_time = MPI_WTIME()
  do k=ks,ke
    aline = H(:,k)
    call mpi_gather(aline,JE,MPI_REAL,global_line,JE,MPI_REAL,0,...)
    if (mype.eq.0) then
       write(15,rec=(cnt-1)*ke+k) (global_line(j),J=JS,JE*NPES)
    end if
  end do
  call MPI_BARRIER(MPI_COMM_WORLD,ierr)
  write_time = MPI_WTIME()-write_time
  sum = sum + write_time
  if (mype.eq.0) print *,'Frame ',cnt,'time was ',write_time,'seconds'
  end if
end do
```

MPI Gather


160
140
120
100
80
60
40
20
0
116
142
140
145
144
142
142
148
145
141
131
142
134
138
128
143
135
143
128
139
138
120
2
3
4
5
6
8
4
4.5
5
6
7.5
4
6
6.67
8
2
3
5
6
6.25
6.75
7.5
Tasks/Node

5th Version - MPI_File (1 of 2)

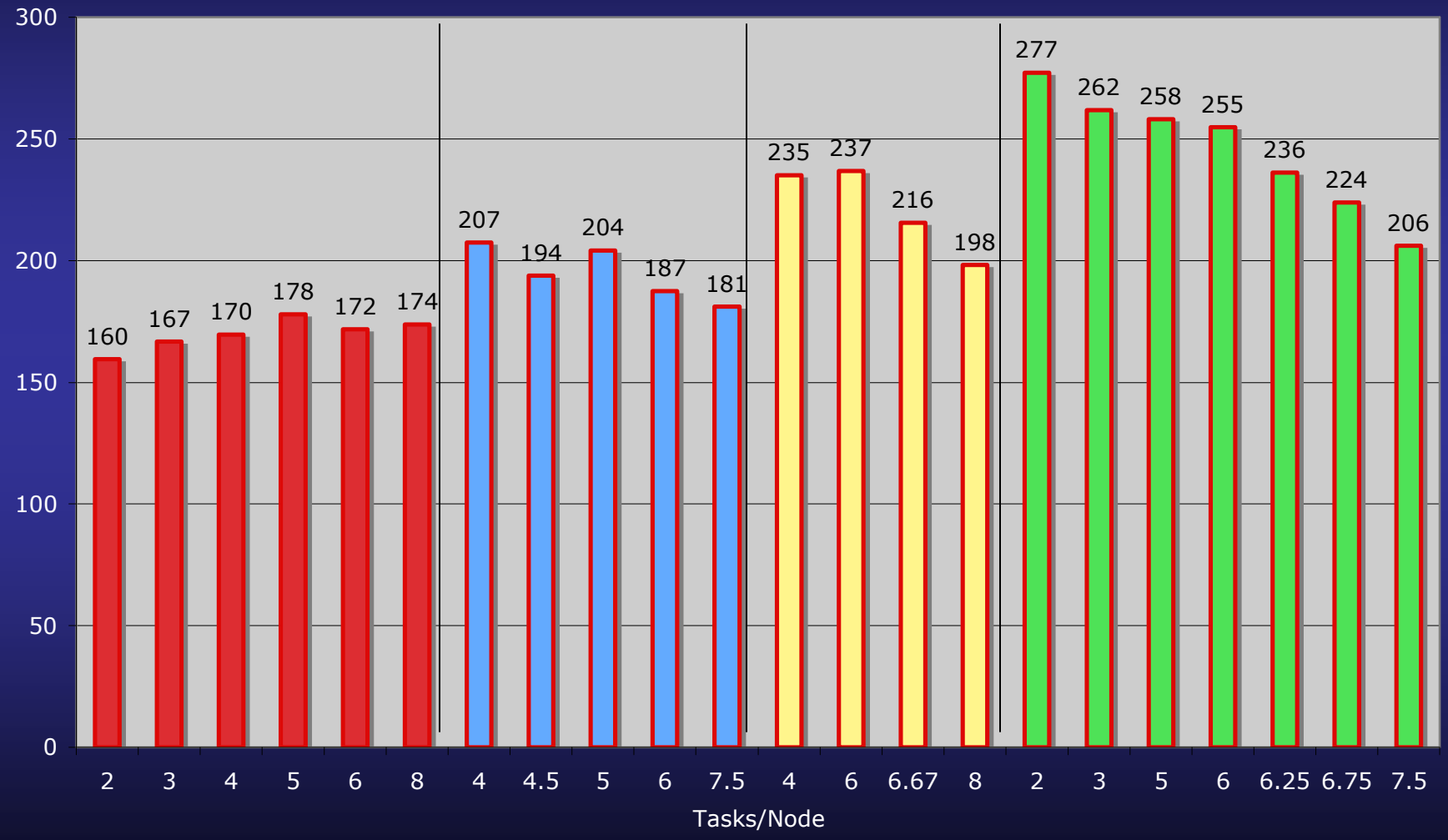
- First Create the Data Type

```
call MPI_TYPE_VECTOR(KE,JE,GLOBAL_JE,MPI_REAL,   &
                                    filetype,ierr)
call MPI_TYPE_COMMIT(filetype,ierr)
call MPI_FILE_OPEN(MPI_COMM_WORLD, 'testsrc4min.bin', &
          MPI_MODE_WRONLY+MPI_MODE_CREATE,      &
          MPI_INFO_NULL, thefile, ierr)
```

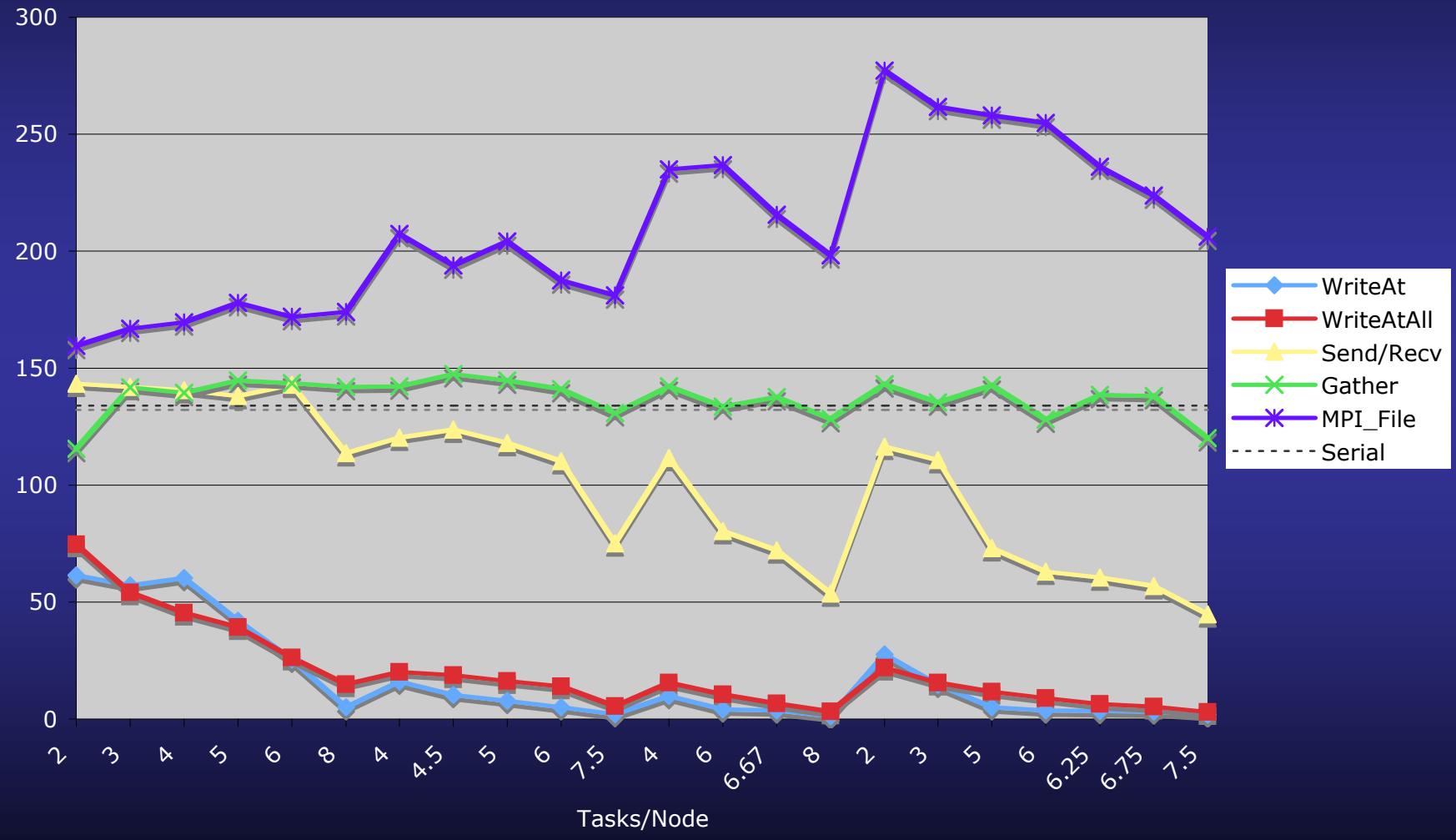
5th Version - MPI_File (2 of 2)

```
filesize = JE*KE*data_size*NPES
do cnt=1,60
  write_time = MPI_WTIME()
  call MPI_FILE_SET_VIEW(thefile,myoffset,MPI_REAL,filetype,"native",  &
                         MPI_INFO_NULL,ierr)
  call MPI_FILE_WRITE_ALL(thefile,H,JE*KE,MPI_REAL,                    &
                         MPI_STATUS_IGNORE,ierr)
  write_time = MPI_WTIME()-write_time
  sum = sum + write_time
  if (mype.eq.0) print *,'Frame ',cnt,'time was ',write_time,'seconds'
  myoffset = myoffset + filesize
end do
```

MPI_File


300
250
200
150
100
50
0
160
167
170
178
172
174
207
194
204
187
181
235
237
216
198
277
262
258
255
236
224
206
2
3
4
5
6
8
4
4.5
5
6
7.5
4
6
6.67
8
2
3
5
6
6.25
6.75
7.5
Tasks/Node

Overall Comparison of IO Tests



MPI-2 One-Sided Communication

- **Since CAF is all one-sided, seemed natural to replace with MPI-2 One-Sided communication**
  - Rewrite was anything but natural…
    - Took several weeks to get it working
    - Speed was on the order of 1/100th of CAF version on CRAY X1 (while using WRITE_AT_ALL for the IO)

MPI2 One-Sided Testing

- **Four basic data transfers were explored trying both fence and lock for synchronization methods**

1) Get an entire array:

```
{sync}
call MPI_GET(tmp,M*N,MPI_INTEGER,PREV,disp,M*N,MPI_INTEGER,cell_win,ierr)
{sync}
```

2) Put an entire array:

```
{sync}
call MPI_PUT(tmp,M*N,MPI_INTEGER,PREV,disp,M*N,MPI_INTEGER,cell_win,ierr)
{sync}
```

Test Performed (2)

3) Put single values into an array:

```
DO i=1,N
   DO j=1,M
      disp = (i-1)*M+j-1
      k = (i-1)*M+j-1
     {sync}
      call MPI_PUT(k,1,MPI_INTEGER,PREV,disp,1, &
                          MPI_INTEGER,cell_win,ierr)
      {sync}
   END DO
END DO
```

Test Performed (3)

4) Put single values from an array into an array:

```
{sync}
DO i=1,N
   DO j=1,M
      disp = (i-1)*M+j-1
      call MPI_PUT(tmp(j,i),1,MPI_INTEGER,PREV,disp,1, &
                           MPI_INTEGER,cell_win,ierr)
   END DO
END DO
{sync}
```

MPI-2 Initial Testing Results

- **Fence is faster than lock**
- **However, fence is NOT one-sided!**
- **Synchronization takes a long time**

METHOD	WIN_FENCE	WIN_LOCK
Get an array	111 MB/s	68 MB/s
Put an array	1480 MB/s	1460 MB/s
Put single values to array	0.05 MB/s	0.02 MB/s
Put an array into an array	4.0 MB/s	3.9 MB/s

Modification History

- **Tenfold increase in speed due to switching to bulk MPI-1 communication methods**

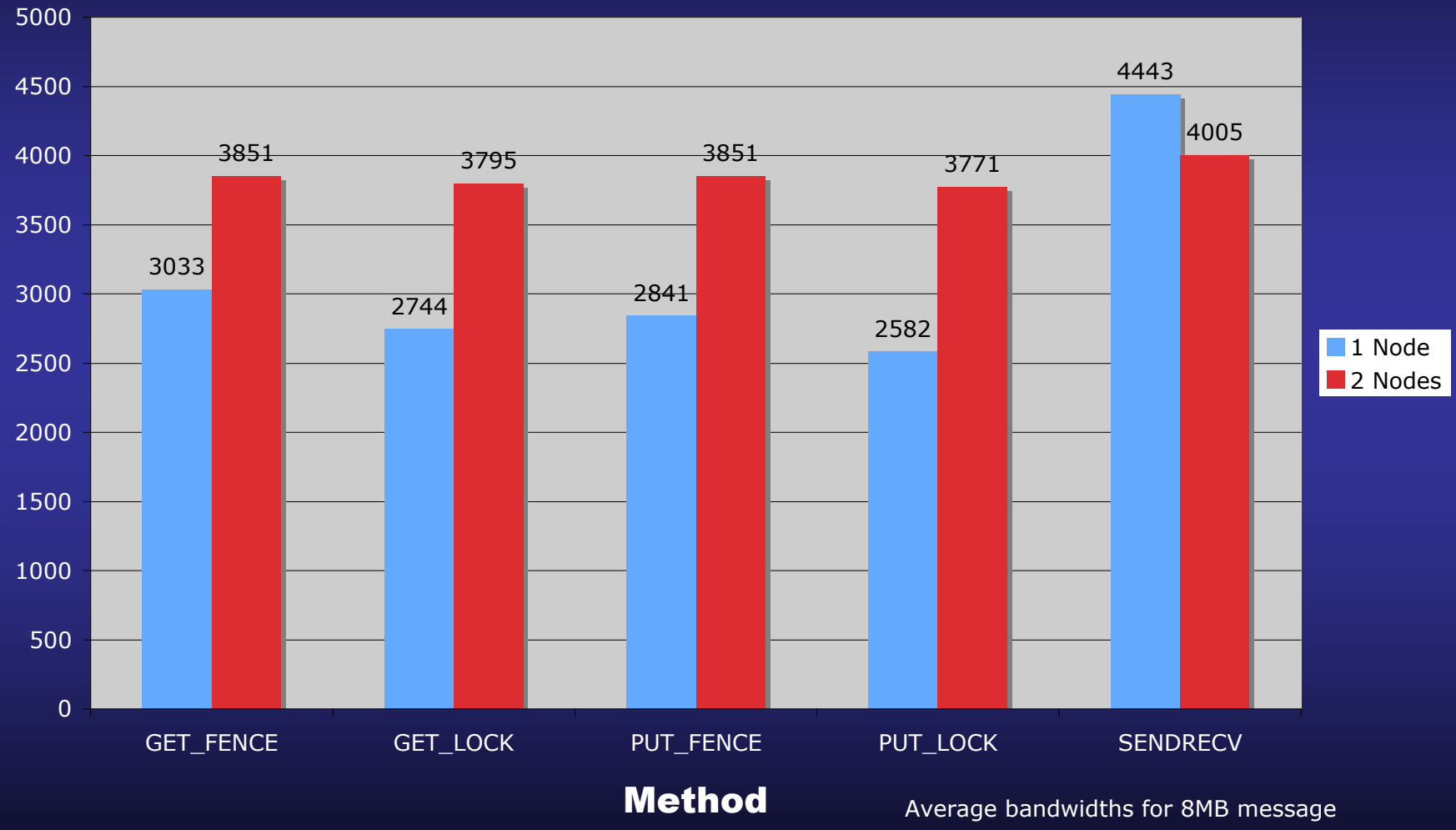
METHOD	Iterations/Minute
All gets/puts	3.63
Changed velocityx to sendrecv	9.33
Changed contx to sendrecv	13.9
Changed velocityy to sendrecv	32.7
Removal of extraneous barriers	37.3

Additional MPI-2 Testing

- **Simple BW Test Kernel**

```
do while (n.le.MAX_LEN)
   tmp = MPI_Wtime()
   do i=1,REPEATS
        call MPI_WIN_FENCE(0,thewin,ierr)
        call MPI_GET(rbuf,n,MPI_INTEGER,from,disp,n,MPI_INTEGER,thewin,ierr)
        call MPI_WIN_FENCE(0,thewin,ierr)
   end do
   tmp = (MPI_Wtime() - tmp)/real(REPEATS)        ! Time per call
   bw = 2.0*real(n)*kind(sbuf)/tmp/1000000.0      ! bw - unidirectional in MB/sec
   call MPI_Barrier(mpi_comm_world,ierr)
   print *,'       SIZE:',n,' PE:',mype,' BW = ',bw,' Time = ',tmp
   n = n * mult
end do
```

MPI Point-to-Point



Average bandwidths for 8MB message

Global Tsunami Code in MPI

- **Ended up removing nearly all MPI-2 one-sided communication**
  - Only 3 GETs and 4 PUTs left in main loop of code
    - Using locks NOT fences
  - Used sendrecv calls to transfer boundary cells wherever they were needed
- **For last science run, used MPI_Gather version of IO**
  - Will certainly implement File_view version next

MPI2 Windows with Locks

```
call MPI_WIN_CREATE(uo,size,data_size,MPI_INFO_NULL,MPI_COMM_WORLD,&
                    uo_win,ierr)

!--- fill iktmp1 with cell(js,:) array from NEXT PE ----------------------
ikbuff=cell(js,:)
call MPI_SENDRECV(ikbuff,KE,MPI_INTEGER,PREV,M*100+1, &
                  iktmp1,KE,MPI_INTEGER,NEXT,M*100+1, …)

!--- fill ktmp2 with slo(js,:) array from NEXT PE ----------------------------
kbuff=slo(js,:)
call MPI_SENDRECV(kbuff,KE,MPI_REAL,PREV,M*100+2, &
                  ktmp2,KE,MPI_REAL,NEXT,M*100+2, …)

!--- fill iktmp2 with cell(js+1,:) array from NEXT PE ------------------
ikbuff=cell(js+1,:)
call MPI_SENDRECV(ikbuff,KE,MPI_INTEGER,PREV,M*100+3, &
                  iktmp2,KE,MPI_INTEGER,NEXT,M*100+3, …)
```

MPI2 - Windows with Locks (2)

```
DO K=KS,KE
   IF (CELL(JE,K).EQ.0.AND.iktmp1(K).EQ.1) THEN
      IF ((ktmp2(K) + H(JE,K)-ETO(JE,K)).GT.EPSILON) then
         D(JE,K)= ktmp2(K) + H(JE,K)-ETO(JE,K)
         CELL(JE,K) = 1
         CELLX(JE,K) = 1
         IF(iktmp2(K).EQ.1)THEN
            disp = (K-1)*JE+(JS+1)-1
            call MPI_WIN_LOCK(MPI_LOCK_SHARED,NEXT,0,uo_win,ierr)
            call MPI_GET(ftmp,1,MPI_REAL,NEXT,disp,1,MPI_REAL,uo_win,ierr)
            call MPI_WIN_UNLOCK(NEXT,uo_win,ierr)
            disp = disp-1
            call MPI_WIN_LOCK(MPI_LOCK_SHARED,NEXT,0,uo_win,ierr)
            call MPI_PUT(ftmp,1,MPI_REAL,NEXT,disp,1,MPI_REAL,uo_win,ierr)
            call MPI_WIN_UNLOCK(NEXT,uo_win,ierr)
```

MPI2 - Windows with Locks (3)

```
            ELSE
               ftmp=-SQRT(G*D(JE,K))
               disp=(K-1)*JE+JS-1
               call MPI_WIN_LOCK(MPI_LOCK_SHARED,NEXT,0,uo_win,ierr)
               call MPI_PUT(ftmp,1,MPI_REAL,NEXT,disp,1,MPI_REAL,uo_win,ierr)
               call MPI_WIN_UNLOCK(NEXT,uo_win,ierr)
            END IF
         END IF
      END IF
   END DO
```

Conclusions

- **In the end, we got the Pacific Tsunami simulation accomplished**
  - Final run used 1-minute data posting over portion of globe (140 Million grid cells)
  - Used 30 8-way Power4 nodes which took 17 hours 15 minutes to complete the 20 hour simulation

MPI-IO

- Can be VERY bad if used incorrectly
- Can be VERY good if used correctly
- Use of MPI-2 file views is imperative to performance
- Good old MPI_GATHER is still your friend!

MPI-2 One-Sided

- Using FENCE is NOT one-sided
- Using LOCKS is one-sided but slower
- Will not use these again if I can help it!
- **We should all (as programmers) push vendors very HARD for Co-Array Fortran!!!**

Maximum Sea Level

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

Acknowledgements

- Dr. Zygmunt Kowalik, UAF/IMS
- Stephanie McAllister, ARSC Intern
- Dr. David Cronk, UTK/PET
- Dick Treumann, IBM